**Docker Cheat Sheet**

1. **Container Lifecycle**

# Run a container in detached mode (-d) and map port 8080 to container's 80

*docker run -d -p 8080:80 --name myapp nginx:alpine*

# Start, stop, and restart the container

*docker start myapp*

*docker stop myapp*

*docker restart myapp*

# Remove a container (must be stopped first)

*docker rm myapp*

# Run a container with CPU and memory limits

*docker run -it --cpus="1.5" --memory="512m" ubuntu*

# Run a container that auto-removes itself after exit

*docker run --rm redis*

# Execute a command inside the running container

*docker exec -it myapp sh # Change "sh" to "bash" if using Ubuntu/Debian*

1. **Image Management**

# Build an image (specify Dockerfile location with ".")

*docker build -t myimage:1.0 .*

# List images

*docker images*

# Remove unused images

*docker image prune -a*

# Save image for offline use

*docker save -o myimage.tar myimage:1.0*

# Load image from tar file

*docker load -i myimage.tar*

1. **Networking**

# Create a network

*docker network create mynet*

# Connect containers to the created network

*docker run -d --net=mynet --name app1 myimage*

*docker run -d --net=mynet --name app2 myimage*

# Inspect ports (Incorrect command in your case, corrected below)

*docker port app1 # Check ports for app1*

*docker port app2 # Check ports for app2*

1. **Volumes & Data**

# Create a volume

*docker volume create myvol*

# Mount host directory as a volume

*docker run -v /host/path:/container/path nginx*

# Mount named volume (myvol) to a container

*docker run -v myvol:/container/path nginx*

# Backup volume data to current directory

*docker run --rm -v myvol:/data -v "$(pwd)":/backup ubuntu tar cvf /backup/backup.tar /data*

1. **Docker Compose**

**docker-compose.yml**

version: '3.8'

services:

web:

image: nginx:alpine

ports:

- "80:80"

volumes:

- ./html:/usr/share/nginx/html

depends\_on:

- db # Ensures the database starts before the web service

db:

image: postgres:13

environment:

POSTGRES\_USER: user # Best practice: Define a user

POSTGRES\_PASSWORD: example

POSTGRES\_DB: mydatabase # Best practice: Define a database

volumes:

- db-data:/var/lib/postgresql/data # Persistent storage

volumes:

db-data: # Named volume for PostgreSQL persistence

**Commands**

# Start services in detached mode

*docker-compose up -d*

# View logs (follow mode)

*docker-compose logs -f*

# Scale the web service to 3 instances

*docker-compose up -d --scale web=3*

1. **Debugging & Maintenance**

# Monitor resource usage of running containers (CPU, Memory, Network, etc.)

*docker stats*

# Show running processes inside a container

*docker top myapp*

# View real-time logs of a container

*docker logs -f myapp*

# Inspect detailed metadata and configuration of a container

*docker inspect myapp*

# Monitor real-time Docker events (container starts, stops, network changes, etc.)

*docker events*

# Show filesystem changes made inside a container (compared to its image)

*docker diff myapp*

1. **Security**

# Run a container as a non-root user (UID:GID)

*docker run --user 1000:1000 myimage*

# Scan an image for vulnerabilities (Ensure Docker Scan is installed)

*docker scan myimage*

# Drop all capabilities and allow only necessary ones (e.g., for binding to privileged ports)

*docker run --cap-drop=ALL --cap-add=NET\_BIND\_SERVICE nginx*

1. **CI/CD Integration**

# Stage 1: Build the application

*FROM node:16 AS builder*

# Set working directory

*WORKDIR /app*

# Copy package.json and package-lock.json separately for better caching

*COPY package.json package-lock.json ./*

# Install dependencies

*RUN npm install --only=production*

# Copy the entire source code after dependencies are installed

*COPY . .*

# Build the application

*RUN npm run build*

# Stage 2: Serve using Nginx

*FROM nginx:alpine*

# Copy built application from builder stage

*COPY --from=builder /app/dist /usr/share/nginx/html*

# Expose port 80

*EXPOSE 80*

# Start Nginx

*CMD ["nginx", "-g", "daemon off;"]*

1. **Kubernetes**

# Generate Kubernetes YAML from Docker Compose

*kompose convert -o k8s-manifest.yaml*

# Start Minikube

*minikube start*

# Apply the generated Kubernetes manifests

*kubectl apply -f k8s-manifest.yaml*